django-user-connections Documentation

Release 0.0.1

Troy Grosfield

January 27, 2014

Contents

CHAPTER 1

Intallation

Install the app:

```
pip install django-user-connections
```

CHAPTER 2

Configuration

2.1 Extending the UserConnection Model with Model Mixin Hooks

There are times when a generic 3rd party model doesn't quite give you all the functionality you'd like. Things like project specific settings or adding helper functions like:

```
def get_absolute_url(...)
```

This app give you the ability to add a mixin to the UserConnection model to alter it's behavior.

2.1.1 Creating the Model Mixin

Create the mixin you want to apply to the UserConnection model:

```
# my_user_connection_app/models.py
from django.db import models

class AbstractUserConnectionMixin(models.Model):
    """The abstract user connection model to add functionality to the
    UserConnection's model.
    """

    class Meta:
        abstract = True

    def get_absolute_url(self):
        return reverse('my_user_connection_url_name', args=[self.id])

    def my_new_method(self):
        # do something with the user connection object
        return 'works'
```

2.1.2 Configuring the Mixin

In your django settings.py file, include the `USER_CONNECTION_MODEL_MIXIN` that points to your user connection model mixin:

```
USER_CONNECTION_MODEL_MIXIN = 'my_user_connections_app.AbstractUserConnectionMixin'
```

2.1.3 Using the New Model

Now that the mixin has been created and configured, let's use it:

```
>>> from django_user_connections.models import UserConnection
>>> n = UserConnection()
>>> n.my_new_method()
'works'
```

2.2 Using a Custom Model Manager

There are also times when you want to customize a model manager, but don't want to create another concrete implementation or proxy model. Here's how you extend or override the object manager model.

2.2.1 Creating the Model Manager

Create the manager you want to user for the UserConnection model:

```
# my_user_connection_app/managers.py
from django_user_connections.managers import UserConnectionManager


class MyUserConnectionManager(UserConnectionManager):
    """Manager for overriding the UserConnection's manager."""

    def my_new_manager_method(self):
        return 'works'
```

2.2.2 Configuring the Manager

In your django settings.py file, include the `USER_CONNECTION_MANAGER` that points to user connection manager you want to use for the project:

```
USER_CONNECTION_MANAGER = 'my_user_connections_app.managers.MyUserConnectionManager'
```

2.2.3 Using the New Manager

Now that the manager has been created and configured, let's use it:

```
>>> from django_user_connections.models import UserConnection
>>> n = UserConnection.objects.my_new_manager_method()
'works'
```

2.3 Extend the Model

If all this configuration still isn't to your liking, then you can simply extend the AbstractUserConnection model:

```python
# my_user_connection_app/models.py

from django_user_connections.models import AbstractUserConnection

class MyUserConnection(AbstractUserConnection):
    """Your concrete implementation of the user connection app."""
    # Do your stuff here
```

CHAPTER 3

Examples

Below are some basic examples on how to use django-user-connections:

```
>>> from django.contrib.auth import get_user_model
>>> from django_user_connections.models import UserConnection
>>>
>>> User = get_user_model()
>>> user_1 = User.objects.create_user(username='hello')
>>> user_2 = User.objects.create_user(username='world')
>>>
>>> conn = UserConnection.objects.create(created_user=user_1,
...                                      with_user=user_2)
>>> conn.status
'PENDING'
>>> user_connection = UserConnection.objects.get_for_users(user_1=user_1,
...                                                        user_2=user_2)
>>> conn == user_connection
True
```

CHAPTER 4

API Reference

4.1 Models

4.2 Mixins

4.2.1 Forms

4.2.2 Views

4.3 Managers

4.4 Template Tags

django_user_connections.templatetags.user_connection_tags.**get_connected_user**(*user_connection*, *auth_user*)

Gets the user the authenticated user is connected with.

4.5 Constants

class django_user_connections.constants.**Status**

The different status's a user connection can be in.

Field ACCEPTED an accepted and current user connection

Field DECLINED a declined user connection. This connection was never in an ACCEPTED state, or active.

Field PENDING the user connection is pending and waiting on a response from the user.

Field INACTIVE represents a user connection for two users that was once accepted and is no longer.

4.6 Helper Methods

django_user_connections.**get_user_connection_model**()

Return the UserConnection model that is active in this project.

This is the same pattern user for django's "get_user_model()" method. To allow you to set the model instance to a different model subclass.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

d